C＋＋による天体の運動のシミュレーション

下　村　忠　行

1. はじめに

われわれにとって地球はきわめて広大な存在であり，地球が球形であることを実感することはほとんどないと言ってよい。また，太陽が早朝に東の空に現れ，夕方西の空に沈むという日常の経験からすると，地球が自転しているということも考えにくいことで，むしろ，地球のまわりを太陽が回っているという，天動説の方がわかりやすいであろう。しかし，小学校，中学校における天体の運動に関する種々の実験・観察や指導を通して，母なる地球が太陽系の一員として，太陽のまわりを自転しながら公転しているというモデルが，納得しているか否かは別として，われわれの頭の中に刻みこまれている。

太陽を中心とする諸惑星の運動や惑星の周囲を回る衛星の運動，さらには地球の自転運動について，地上からの観測で理解することは容易ではない。このような複雑な相対運動を考える場合には，その運動を表す座標系として考えやすい単純なものを選び，目的とする座標系へと順次変換して考えていくようにすれば理解しやすい。そのためには，目的とする運動をコンピュータ・シミュレーションによって，単に数量化するだけでなく視覚的にとらえながら考えるようにするのが効果的であり，教育的に重要なことであると考える。

このようなシミュレーションのプログラム作成において，プログラム言語C＋＋は諸現象を階層的にクラスに分けて記述するので，相対的な運動を表わす場合に考えやすく，記述もかなり容易になると考えられる。したがって，太陽，惑星，惑星の周囲を回る衛星と３つの階層に整然と分けられる太陽系内の諸運動を調べるには好都合である(図１)。そこで，プログラム言語C＋＋を用いて，太陽系における惑星や衛星の運動を調べるシミュレーション・プログラムをいくつか作成し，検討したので報告する。ここで報告するのは，一つは，コインのまわりを回るもう一枚のコインの運動の問題である。これは惑星や衛星の自転・公転につながる問題である。さらに，月は太陽のまわりどのように回っているかを調べるシミュレーションついて，および，地球自転の証拠としてのフーコー振り子の運動の原理を視覚的に考察していくためのシミュレーションについてであり，いずれも基礎的な諸現象を試行錯誤しながら把握していくための教材例として作成・検討したものである。

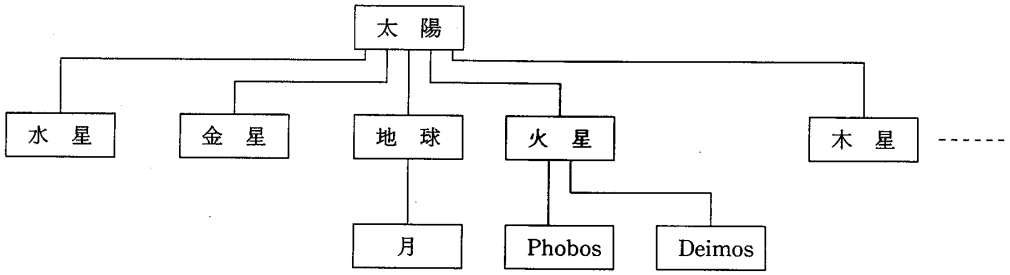

図1　太陽系の階層構造

2. C＋＋言語によるプログラミング

C＋＋はオブジェクト指向プログラム言語であり，構造化プログラミングの考え方を取り入れたC言語をさらに発展させたものである。C＋＋の詳細については専門書ゆずり，太陽系における惑星や衛星の運動のシミュレーションを例に，簡単にプログラムの構造について説明する。

C＋＋では自然界の事象をシミュレートしたものをオブジェクトといい，対象とする事象を目的に応じていくつかのオブジェクトに分解して，それぞれのオブジェクトにメソッド（関数）と属性を割り当てていくという手順でプログラムの作成を行う。このようにオブジェクト単位で作業を行うので，オブジェクト指向プログラミングというのである。また，オブジェクトをグループ化するためにクラスを定義する。クラスは対応するカテゴリに属するオブジェクト群からなる。たとえば太陽系をシミュレートするときには，図1に示すように，中心となる太陽クラスを頂点に，水星，金星，地球，火星，木星等々の惑星をそれぞれ太陽クラスの下の階層のクラスとし，さらに地球をまわる月は地球の下の階層のクラス，火星の2つの衛星は火星の下の階層のクラスというように，階層構造で太陽系を考えるのである。

また，メソッドや属性の初期化（オブジェクトの初期化）を行うメソッドをコンストラクタという。コンストラクタはオブジェクトが作成されたときに，最初に起動される。

太陽系における地球と月の運動を考える場合，太陽クラスは太陽の中心を原点とする座標系で地球や月の位置を求めるメソッドなどや，太陽の質量，直径，万有引力定数等々の属性を必要に応じてもたせる。太陽クラスの下位クラスである地球クラスにおいては，地球中心を原点とする座標系で月の位置を求めるメソッド，直径，質量，公転の半径や角速度，自転周期などの属性が必要である。また，地球の下位クラスとなる月クラスは，月面上での運動等を考えるのでなければ，月の直径，質量，公転の半径および角速度，自転周期などの属性を与える。そして，メインプログラムによって，太陽－地球－月の各クラスの階層を上下しながらプログラミングしていくのである。

3. 2枚のコインの問題

(1) 問題の考え方

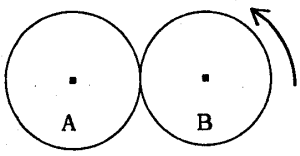

図2

同じ大きさ2枚のコインA，Bを用意し，図2のように固定したコインAの縁に沿って，もう1枚のコインBを滑らせないようにして回転させる。このとき，「BがAのまわりを1回転すると，B自身は何回転するか」という問題は，よく知られた興味深い問題である。この問題は，地球のまわりを回る月の運動や地球の公転，自転などの問題に直結するので，まず，このコインの問題について考察することにする。

直感的には，2つのコインの円周の長さは等しいのだから，1回転しかしないように思いがちである。しかし，実験してみれば明らかなように，実際には2回転する。なぜ2回転するのかについては，いろいろな考え方，説明の仕方があろう。

ひとつの考え方として，図3のように，コインBが直線上を回転する場合(a)と，コインAの周囲を回転する場合(b)を比較してみるとよい。円周上を回転する場合は，直線上を回転する場合に比べると，図からも明らかなように，回転角は2倍である。したがって，コインBが直線上を1回転すると1円周分進むのに対し，コインAに沿って1周する（1円周分進む）場合はその2倍回転する，すなわち2回転するのである。コインBの運動を直線上の運動と考えてしまうために，1回転と錯覚するのである。

また，図3において，コインBの(a)および(b)の場合の回転半径を比べてもよい。回転半径は，(b)は(a)の2倍であるから，直線上の運動に比べて，円周上の運動の回転数は明らかに2倍になる。

(2) C＋＋言語によるシミュレーション

このほかにも種々の考え方があろうが，この問題はコンピュータ・シミュレーションにより視覚化して理解するのが，手軽で有効な方法の一つと思う。そこで，プログラム言語C＋＋により，この問題のシミュレーションを行うプログラムの作成について考える。

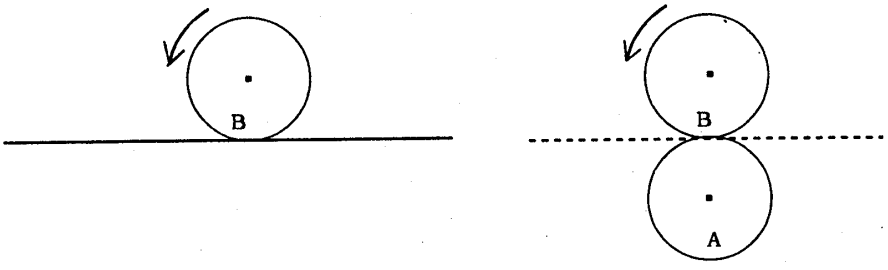

図3 (a)　(b)

クラスとして，図4に示すように，階層構造をなすCoinAおよびCoinBの2つのクラスを考える。すなわち，上位クラスであるCoinAクラスは，図5(a)に示すように，コインAの中心を原点とする座標系において，コインAの周囲を回るコインBの運動を表わすオブジェクトのグループである。すなわち，コインAを中心とする座標系で観察した場合のコインBの運動に関するものある。また，CoinBクラスはCoinAクラスの下位クラスであり，図5(b)に示すように，コインBがコインAに接する点を原点とし，接線とそれに直交する直線を座標軸として，接線上を転がるコインBの運動を表すオブジェクトのグループである。すなわち，コインBが直線上を回転する場合の運動に関するものである。CoinBクラスによって，各時刻におけるコインBの自転による回転角および移動距離を求めることができる。また，これは，コインAの半径を極端に大きくし場合のコインBの運動と考えることもできる。たとえば，地平面上を転がる球の運動である。

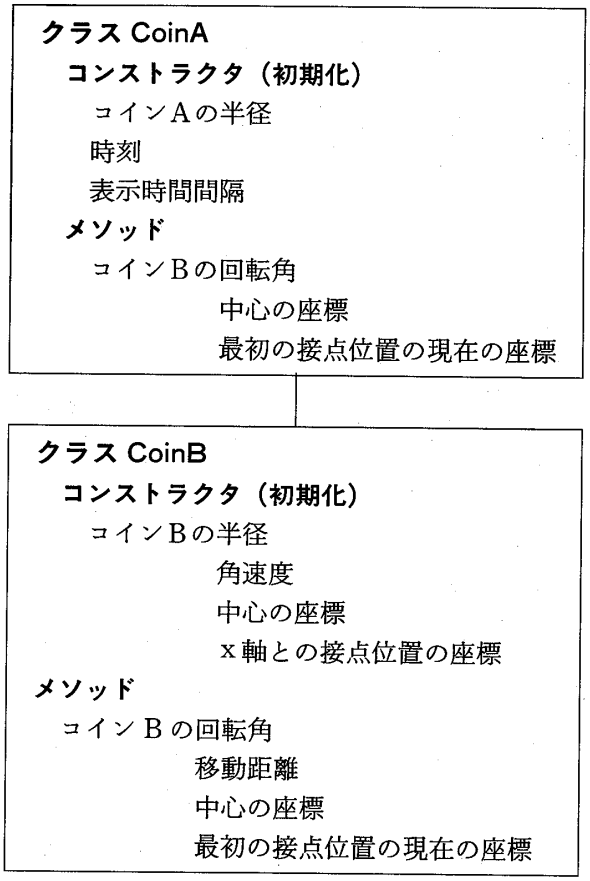

図4　2枚のコインクラス構造

CoinAクラスにおいては，あらかじめ設定すべき属性としてコインAの半径，時刻，表示時間間隔などがある。メソッドとしては，クラスCoinBより求められるコインBの移動距離からコインAのまわりのコインBの回転角を求めるメッソド，コインAの座標におけ

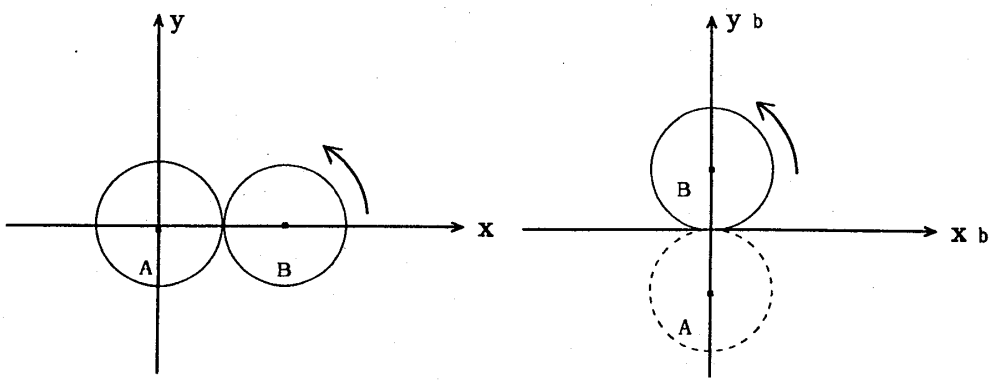

図5（a）　(b)

```
/* coin_program */
#include <stdlib.h>
#include <iostream.h>
#include <math.h>
#include <stdio.h>
#define  PAI   3.141593
#define  D_R   (PAI / 180.)
#define  R_K   1. / (2.* PAI)

class CoinA                    // CoinAクラス
{
public:
    double  tnow, dt;          // 時刻、表示時間間隔
    double  ra, sa;            // ra：ｺｲﾝAの半径, sa：ｺｲﾝAのまわりのｺｲﾝBの回転角
    double  xbc, ybc, xbg,ybg; // xbc,ybc：ｺｲﾝBの中心の座標, xbg,ybg：スタート時の接点の座標
    CoinA()                    // CoinAのコンストラクタ
    {
        ra = 1.0;  tnow = 0.0;  dt = 1.0;   // ｺｲﾝAの半径,時刻,表示時間間隔の設定
    }
    double getsa(double b_x)   // ｺｲﾝAのまわりのｺｲﾝBの回転角saを求める
    {
        sa = -b_x / ra;
        return sa;
    }
    void pos_a_b(double rb, double bww)  // ｺｲﾝBの中心の座標およびスタート時の接点の座標を求める
    {
        xbc = (ra + rb) * cos(sa);
        ybc = (ra + rb) * sin(sa);
        xbg = xbc + rb * cos(PAI + (sa + bww));
        ybg = ybc + rb * sin(PAI + (sa + bww));
    }
    void gett()                // 時刻tnowを求める
    {
        tnow = tnow + dt;
    }
};

class CoinB : public CoinA   // CoinBクラス
{
public:
    double  rb, wb, bww;       // rb：ｺｲﾝBの半径, wb：回転の角速度, bww：回転角
    double  b_x, b_y;          // b_x,b_y：ｺｲﾝBの中心の座標, b_xp,b_yp：スタート時の接点の座標
    double  b_xp, b_yp;
    CoinB()                    // CoinBのコンストラクタ
    {                          // ｺｲﾝBの半径,角速度,スタート時の中心の座標と接点の座標を設定
        rb = 1.0;   wb = 30.0;
        b_x = 0.0; b_y = 0.0;
        b_xp = 0.0; b_yp = rb;
    }
    void dist()                // 時刻tnowにおけるｺｲﾝBの中心およびスタート時の接点の座標を求める
    {
        bww = wb * D_R * tnow;
        b_x = -rb * bww;
        b_xp = b_x + rb * sin(bww);  b_yp = rb - rb * cos(bww);
   }
};

void main()                    // メインプログラム
{
    CoinB  q;
    cout << "¥n          *** コインＢの位置と回転数 ***  ";
    cout << "  ra =" << q.ra << "  rb =" << q.rb << "  wb = " << q.wb << "° " << endl << endl;
    cout << " 時間      xbc       ybc       xbg       ybg     自転回数   公転回数     回転数" << endl;
//      時間間隔1.0ごとに計算して、ｺｲﾝBの中心およびスタート時の接点の座標、および
//      ｺｲﾝBの自転回数、公転回数、回転数（両者の和）を出力
    for(int i =0; i < 13; i++)
    {
        q.dist();
        q.getsa(q.b_x);
        q.pos_a_b(q.rb, q.bww);
        printf("%5.1lf %8.4lf %8.4lf   %8.4lf %8.4lf    ", q.tnow, q.xbc, q.ybc, q.xbg, q.ybg);
        printf("%8.3lf   %8.3lf   %8.3lf¥n", q.bww*R_K, q.sa*R_K, (q.bww+q.sa)*R_K);
        q.gett();
    }
}
```

図６　プログラム例

＊＊＊コインＢの位置と回転数＊＊＊　　ra＝1　rb＝1　wb＝30°

時間	xbc	ybc	xbg	ybg	自転回数	公転回数	回転数
0.0	2.0000	0.0000	1.0000	0.0000	0.000	0.000	0.000
1.0	1.7321	1.0000	1.2321	0.1340	0.083	0.083	0.167
2.0	1.0000	1.7321	1.5000	0.8660	0.167	0.167	0.333
3.0	0.0000	2.0000	1.0000	2.0000	0.250	0.250	0.500
4.0	−1.0000	1.7321	−0.5000	2.5981	0.333	0.333	0.667
5.0	−1.7321	1.0000	−2.2321	1.8660	0.417	0.417	0.833
6.0	−2.0000	0.0000	−3.0000	0.0000	0.500	0.500	1.000
7.0	−1.7321	−1.0000	−2.2320	−1.8660	0.583	0.583	1.167
8.0	−1.0000	−1.7321	−0.5000	−2.5981	0.667	0.667	1.333
9.0	0.0000	−2.0000	1.0000	−2.0000	0.750	0.750	1.500
10.0	1.0000	−1.7321	1.5000	−0.8660	0.833	0.833	1.667
11.0	1.7321	−1.0000	1.2321	−0.1340	0.917	0.917	1.833
12.0	2.0000	0.0000	1.0000	0.0000	1.000	1.000	2.000

図7　出力例

るコインＢの中心の位置の座標とスタート時における直線との接点の現在位置の座標を求めるメソッドを設ける。

CoinB クラスにおいては，あらかじめ設定すべき属性として，コインＢの半径および角速度，コインＢの中心点の座標およびスタート時のｘ軸との接点の座標などがある。メソッドとしては，各時刻におけるコインＢの中心の座標およびスタート時のｘ軸との接点の座標の計算がある。プログラムの実行時に CoinB クラスのオブジェクトを作成されると，上位クラス CoinA，下位クラス CoinB の順に，幹から枝に向かってそれぞれのコンストラクタが自動的に呼び出され，初期化が行われる。

メイン・プログラムにおいては，CoinB クラス，CoinA クラスのメソッドを用いて単位時間ごとに次の計算を行っている。すなわち，直線上を回転する場合のコインＢの回転角を求め，コインＢの移動距離，コインＢの中心の座標およびスタート時における接点の現在の位置を求める。ここで求められた回転角は，コインＢ自身の回転，すなわちコインＢの自転によるものである。また，移動距離からコインＡの中心を原点とする座標系において，コインＢがコインＡの縁にそって移動距離分だけ進んだときの回転角を求める。この回転角は，コインＢ自身は回転することなくコインＡの縁にそってすべった場合の角であり，コインＢの公転によるものである。さらに，この位置でコインＢを自転分だけ回転させ，コインＢの中心の座標およびスタート時における接点の現在の位置の座標を求める。

図6にコインＡの中心を座標系の原点としたときのコインＢの位置の座標および回転数を出力するプログラム例を，図7にその出力例を示す。出力しているのは，コインＡの中心を原点とする座標系におけるコインＢの中心の座標とスタート時における接点の位置の

座標，さらにコインＢの自転回数，公転回数，および回転数（自転分と公転分の和）である。

図７の出力結果からもわかるように，コインＢがコインＡを一回りするとき，コインＢは２回転する。そのうち１回転はコインＢの自転によるものであり，もう１回転はコインＡの周囲を回るコインＢの公転によるものである。

(3) シミュレーションの視覚化

このシミュレーションは，図７のような単なる数値の表示のみによるものではなく，グラフィックスにより視覚的に動きのあるものとして表示しなければならない。また，CoinB クラスからみたコインＢの直線上の運動と，CoinA クラスからみたコインＡの縁にそって回るコインＢの運動の両方をそれぞれウィンドウに表示し，ディスプレイ上で同時に観測できるようにすることが望ましい。そのために，図６のプログラムに図形を描いたり，ウィンドウ表示をするための諸関数を付け加える必要がある。

プログラムリストは省略するが，このようにして得られたシミュレーション結果の画面例を図８に示す。上のウィンドウはコインＡのまわりを回るコインＢの様子であり，コインＢのスタート位置はＡの右側，ｘ軸上からで，反時計回りに進む（図９参照）。太い曲線はスタート時のＡとの接点の位置の動きを示す。この曲線は外サイクロイド曲線である。コインＢの形は15度おきに表示している。下のウィンドウは直線上を進むコインＢの様子で，スタート時のＡとの接点の位置はサイクロイド曲線を描く。なお，図７のデータは，これら２つのウィンドウの背後にある DOS ウィンドウに表示される。

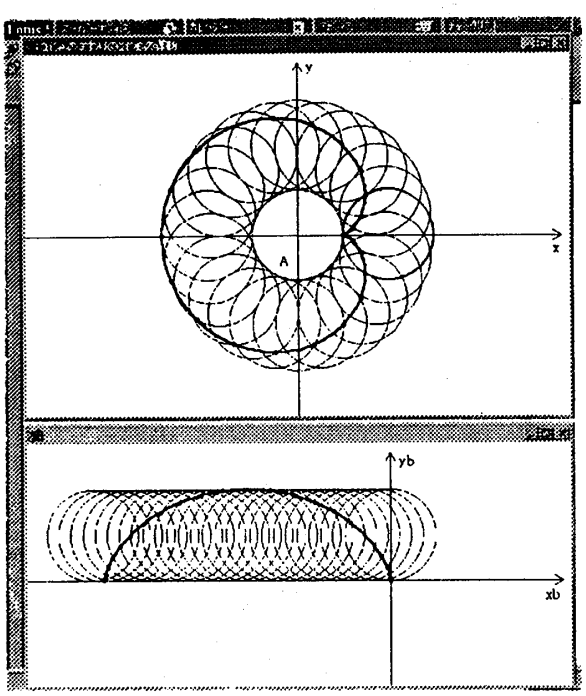

図８　画面例（A，Ｂの半径の比１：１）

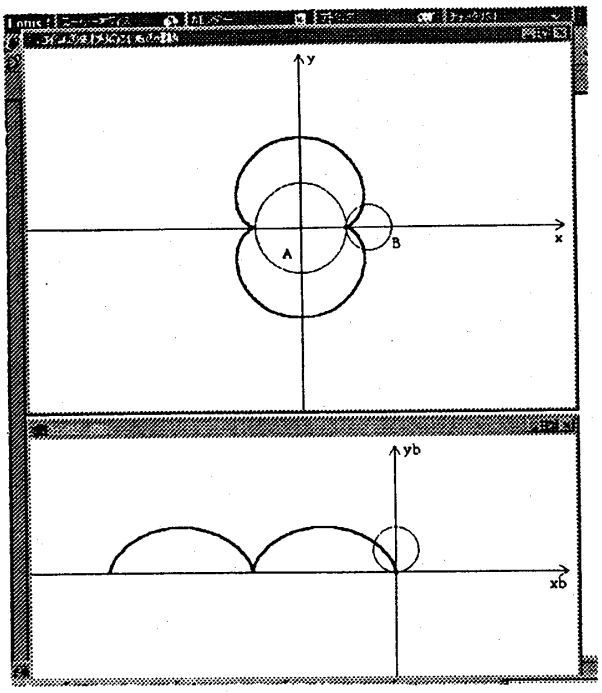

図９　画面例（A，Ｂの半径の比１：1/2）

図９はコインＢの半径をＡの１/２にした場合であり，Ｂは自転２回，公転１回の計３回転することがわかる。なお，コインＢの形はスタート時のみを表示するようにしている。

4．太陽のまわりを回る月の運動

太陽のまわりを回る地球の公転運動は，前の２枚のコインの問題において，コインＡの中心を太陽とし，コインＢの中心を地球として考えればよいわけであり，２枚のコインのプログラムを若干変更するだけで，その軌道をシミュレートすることができる。地球のまわりを回る月の公転運動も，同様である。ここでは，太陽のまわりを回る月の軌道を表すシミュレーションについて考察する。

太陽－地球－月の階層関係を図10のように考える。太陽クラス（クラス Sun）においては，太陽の中心を原点とする座標系からみた地球位置の座標を求めるメソッド，同じく太陽座標系からみた月の位置を求めるメソッド，時間に関するメッソドをおく。地球クラス（クラス Earth）においては，地球の中心を原点とした座標系かみた月の位置を求めるメソッドが必要である。ただし，このメッソドは太陽クラス（クラス Sun）における太陽の中心を原点とする座標系からみた地球位置の座標を求めるメソッドと同じ形なので，これを利用し，実際には記述しない。初期化する属性としては，地球の公転半径および公転角速度がある。月クラス(クラス Moon)においては，月の公転半径および公転角速度を与える。シミュレーションにおいては，太陽，地球，月はいずれも点とみなし，自転については考えない。また，地球および月の公転は円運動とし，座標系は公転面を含む二次元の座標である。

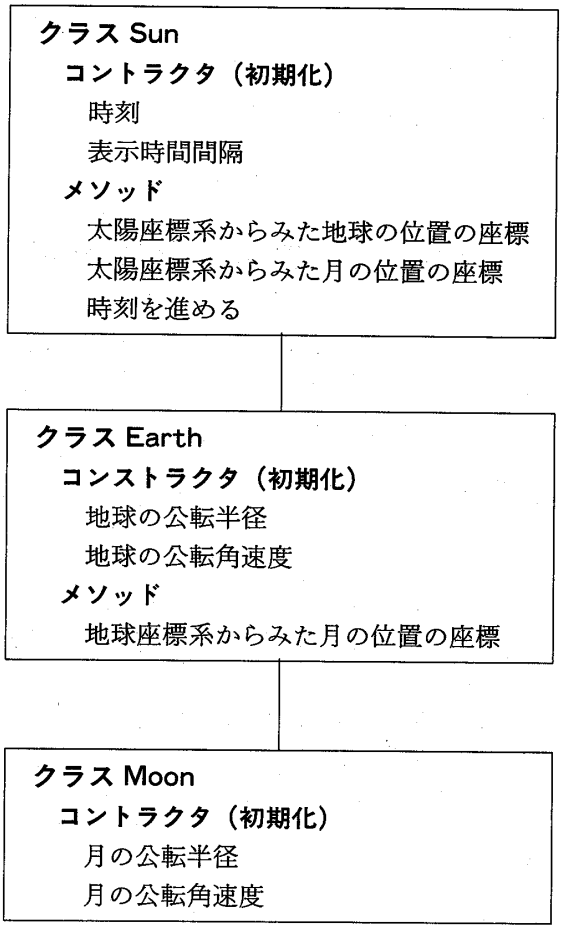

図10　太陽－地球－月のクラス構造

図11はシミュレーションの実行例で，月の公転半径を地球の公転半径の15分の１にとし，月の公転角速度を地球の公転角速度の12倍にとった場合の表示画面例である。左側の大きなウィンドウが太陽のまわりを回る月の運動（小円）と地球の運動（小さな点）を示している。月は太陽のまわりを外サイクロイド曲線を描きながら進む。右下の小さなウィ

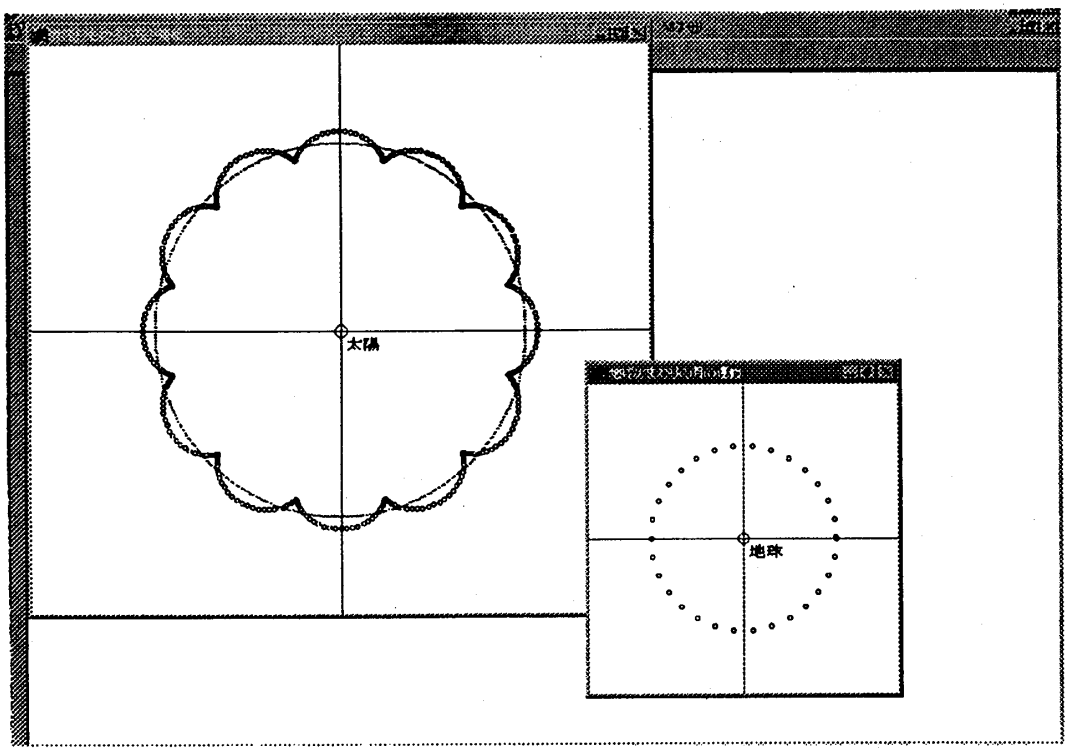

図11　画面例（月の運動）

ンドウは，地球のまわりを回る月の公転運動を表示している。このシミュレーションにおいては，公転角速度の関係は実際に近いが，月の公転半径は実際より数十倍も大きく表している。

5．フーコー振り子のシミュレーション

われわれは地球上に存在していながら，日常，地球が自転していることを体感することはない。地球自転の証拠を簡単に示すことができる装置として唯一とも言えるのが，フーコー振り子である。これは単振り子であるが，その原理は地球の自転を3次元的に考えなければならないことなどから，真の意味での理解は非常に難しい。フーコー振り子の原理に関しては，適切なコンピュータ・シミュレーションを通して把握していくのが，ベストな方法の一つであろう。

赤道上や極点を除いた，いわゆる中緯度における地平面（地面）は，地球の自転運動によって，南北軸を中心にした回転と鉛直軸（地面に垂直な軸）を中心にした回転をしている。もちろんわれわれは気付かないのであるが，赤道上では鉛直軸を中心とした回転がなく，極点では南北軸を中心にした回転がない。地上では，振り子の支点は地面と一緒に動くから，南北軸を中心とする地面の回転は感じない。しかし，振り子は振動面を一定に保つ性質があるので，鉛直軸を中心とする地面の回転は，振動面の回転から測定することが

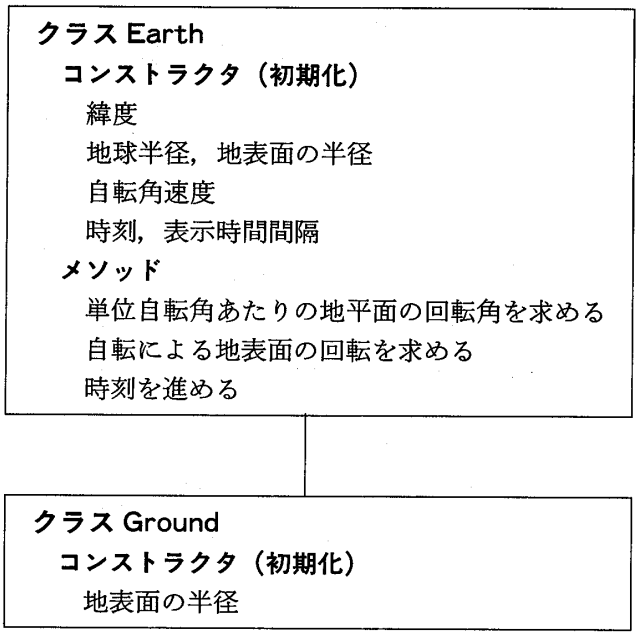

図12　フーコーの振り子のクラス構造

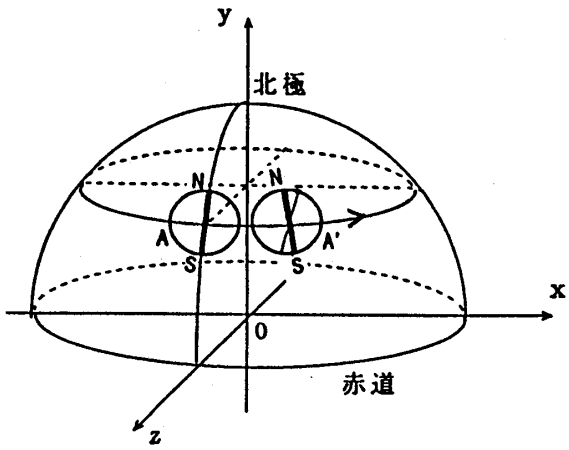

図13　地平面の回転

できる。

詳細は文献(4), (5), (6)等によって頂くことにし，C＋＋言語によるシミュレーションについて考える。図12はクラスの階層ダイアグラムを示したもので，地球クラス(Class Earth)の下に地面クラス（Class Ground）がある。

地球クラスは地球中心を原点とする3次元座標系によって，自転による地平面の動きをシミュレートする。メソッドとしては，地球が微小角だけ自転したときの地平面の鉛直軸のまわりの回転を計算し，単位自転角あたりの回転角を求めるメソッド，自転による地平面の動きを座標変換により求めるメソッド，時間を求めるメソッドなどがある。

自転による地平面の回転角を求めるには，地球の自転を外から観察する地球クラスの方が考えやすい。図13に示す面Aは北半球の緯度 ϕ における半径rの地平面で，y-z面にそって南北線NS(経線)が通っている。地平面の鉛直軸のまわりの回転角を求めるために，面Aを地球の中心に平行移動して考えると，Nの座標 x_1，y_1，z_1は，

$$x_1=0, \quad y_1=r\cos\phi, \quad z_1=-r\sin\phi$$

と表される。この面をy軸を中心にx-z面を微小角 α だけ反時計回りに回転（自転）させると，Nの座標 x_2，y_2，z_2は

$$x_2=r\sin\phi\sin\alpha, \quad y_2=r\cos\phi, \quad z_2=-r\sin\phi\cos\alpha$$

となる。$\alpha \fallingdotseq 0$ であることを考慮すると，自転角がきわめて小さい場合，自転による点Nのx方向の移動は x_2，すなわち $r\sin\phi\sin\alpha \fallingdotseq r\sin\phi \cdot \alpha$ であり，y_2，z_2は変わらないとみなせるので，「自転による鉛直軸のまわりの地平面の回転は，その地点の緯度の正弦に比例す

る」ことを示している。自転角１度に対する鉛直軸のまわりの回転角 a_0は，$\tan^{-1}(x_2/r)=\tan^{-1}(\sin\phi\sin 1°)$である。地平面は自転角１度あたり a_0の割合で，鉛直軸のまわりを反時計回りに回転している。

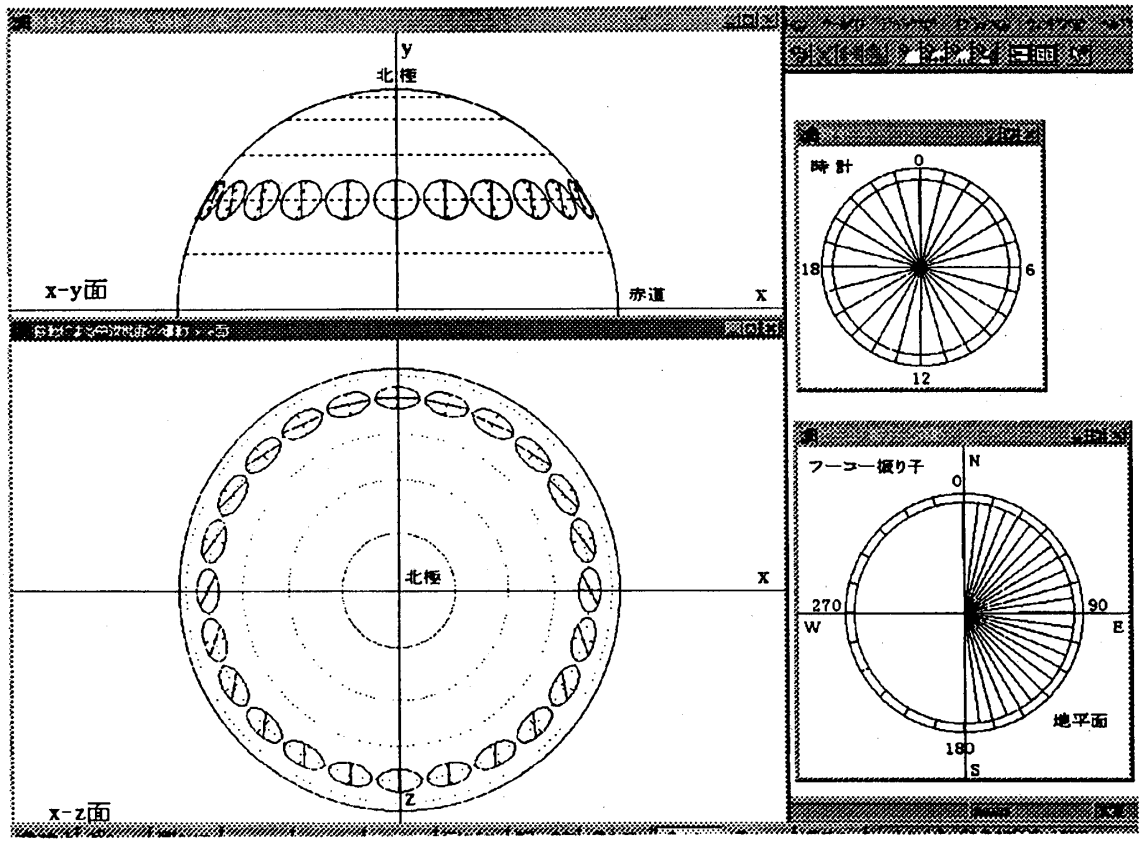

図14　画面例（北緯30度の場合）

＊＊＊フーコーの振り子＊＊＊　北緯30.0度

時間	回転角	理論値	時間	回転角	理論値
0	0.0	0.0	1	7.5	7.5
2	15.0	15.0	3	22.5	22.5
4	30.0	30.0	5	37.5	37.5
6	45.0	45.0	7	52.5	52.5
8	60.0	60.0	9	67.5	67.5
10	75.0	75.0	11	82.5	82.5
12	90.0	90.0	13	97.5	97.5
14	105.0	105.0	15	112.5	112.5
16	120.0	120.0	17	127.5	127.5
18	135.0	135.0	19	142.5	142.5
20	150.0	150.0	21	157.5	157.5
22	165.0	165.0	23	172.5	172.5
24	180.0	180.0			

図15　画面例（北緯30度の場合の DOS ウィンドウ）

地面クラスは，属性として地表面の半径を初期化するだけである。単位時間ごとにフーコー振り子の振動面の動きを出力する仕事は，メインプログラムで行っている。

図14は北緯30度におけるシミュレーション結果の画面で，左上はx-y面から，左下はx-z面から見た地平面の動きを1時間ごとに示している。地平面の点線が南北軸，実線が振り子の振動方向である。右下は地平面上におけるフーコー振り子の振動方向を表している。右上は時計である。

また，図15はシミュレーションより得られた1時間ごとのフーコー振り子の回転角と理論値を示したもので，図14の背後にあるDOSウィンドウに表示されている。

6. おわりに

以上，C＋＋により作成した，相対運動に関連する天文関係のシミュレーションの教材例を紹介した。

プログラミングに関しては，C＋＋は階層構造がはっきりしており，属性の初期化もわかりやすく，比較的しぜんな考え方で作業できるので，構造が複雑で大きなプログラムでも取り組みやすいであろう。ただし，C＋＋が他の言語よりやさしいということはなく，むしろ難しい方に属する。しかし，教育関係のソフト開発において，C＋＋を利用して自然現象をシミュレートする手法，オブジェクト指向プログラミングの手法を体得することは意義あることと思う。教育面での利用としては，C＋＋の特徴を生かして，従来複雑で取り組みにくかった大がかりな教材の作成などが考えられる。

ここで取り上げたシミュレーションは，考える理科教育，科学教育を目指しての教材例であり，高等学校や大学教養程度での利用を意図したものである。

なお，本プログラミングにおいて使用したソフトはBorland C＋＋ version 5.0である。また，ウィンドウ画面の作成やグラフィックスに関しては文献(3)によっている。

文　献

(1) Jeffery M. Cogwell 著 長野文子訳：「かんたんC＋＋」，翔泳社（1994）
(2) Patrick H. Winston 著 鬼頭繁治訳：「ウィンストンのC＋＋」，星雲社（1995）
(3) 平林雅英：「Windows95プログラムを10倍強力に作る（C＋＋版）」，共立出版（1996）
(4) 原島　鮮：「続・物理学覚え書き」，裳華房（1982）
(5) 下村忠行：「フーコー振り子の指導について」，理科研究集録第25号，新潟県高教研理科部会（1986）
(6) 下村忠行：「地球の自転に関する指導について(1)」，暁星論叢第34号，新潟中央短期大学（1994）